From functional programming to multicore parallelism: A case study based on Presburger Arithmetic

Dung, Phan Anh; Hansen, Michael Reichhardt

Published in:
Proceedings of the 23rd Nordic Workshop Programming Theory

Publication date:
2011

From functional programming to multicore parallelism: A case study based on Presburger Arithmetic

Phan Anh Dung and Michael R. Hansen
DTU Informatics, Technical University of Denmark

The overall goal of this work is studying parallelization of functional programs with the specific case study of decision procedures for Presburger Arithmetic (PA). PA is a first order theory of integers accepting addition as its only operation. Whereas it has wide applications in different areas, we are interested in using PA in connection with the Duration Calculus Model Checker (DCMC) [5]. There are effective decision procedures for PA including Cooper's algorithm and the Omega Test; however, their complexity is extremely high with doubly exponential lower bound and triply exponential upper bound [7]. We investigate these decision procedures in the context of multicore parallelism with the hope of exploiting multicore powers. Unfortunately, we are not aware of any prior parallelism research related to decision procedures for PA. The closest work is the preliminary results on parallelism in the SMT-solver Z3 [8] which has the capability of solving Presburger formulas.

Functional programming is well-suited for the domain of decision procedures, and its immutability feature helps to reduce parallelization effort. While Haskell has progressed with a lot of parallelism-related research [6], we choose F# to be able to have explicit control over parallelism on the .NET framework and utilize its option to resort to mutation when optimizing performance.

1 Parallelization of Cooper's algorithm

The algorithm removes quantifiers in the inside-out order using the following transformation [5]:

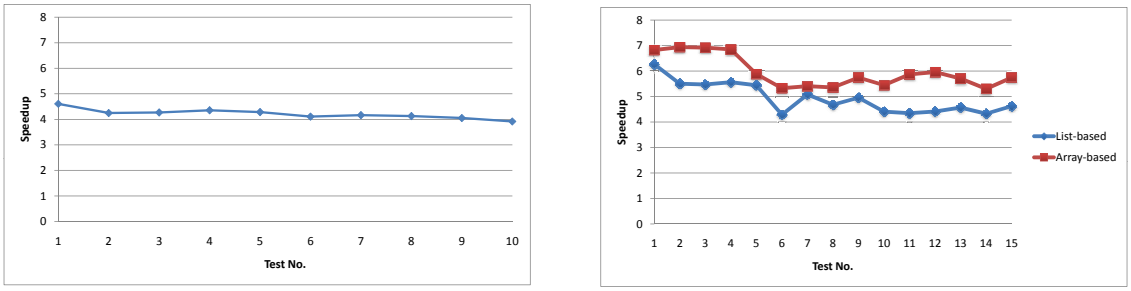
$$\exists\, \mathtt{x}.\ \phi \iff \bigvee_{\mathtt{i}=1}^{\delta} (\phi[\top/\mathtt{ax} < \mathtt{t},\ \bot/\mathtt{ax} > \mathtt{t}] \vee \bigvee_{\mathtt{ax}<\mathtt{t}} \phi[\mathtt{t}+\mathtt{i}/\mathtt{ax}] \wedge \delta' \mid \mathtt{t}+\mathtt{i}) \tag{1}$$

where a > 0, δ is the least common multiple of the coefficients of x in ϕ and δ' is the least common multiple of the divisors in divisibility constraints.

Before parallelization, several optimizations have been considered for Cooper's algorithm. Interestingly, *eliminating blocks of quantifiers* [3] has shown its good performance on the multicore platform. This procedure is superior as quantifiers are distributed into inner formulas for quantifier removal, resulting in manipulation of small data structures which is extremely fast when data is likely to fit in cache. Our preliminary test with several small formulas shows that this optimization leads to $20\times$ performance gain compared to the baseline variant [4].

Cooper's algorithm works with Presburger formulas in negation normal form (NNF). In an ideal case, many formulas need to be resolved simultaneously and we can employ parallelism constructs to utilize multicore powers efficiently. In other cases, parallelism can be extracted from the structure of Cooper's transformation. In Equation 1, denote $\mathtt{B}$ as the number of $\mathtt{ax} < \mathtt{t}$ constraints in ϕ. Despite the symbolic representation of disjunctions, $\mathtt{B}+1$ substitutions should be performed in the formula. As these substitutions are totally independent, we are able to execute them in parallel. The value of $\mathtt{B}$ increases after each elimination step; therefore, the chance of parallelism is ensured.

Degree of parallelism is even more significant if we keep Presburger formulas in disjunctive normal form (DNF). The advantage is the utilization of all available cores for parallel execution and the disadvantage is the explosion of memory usage which could go beyond the capability of the system. We have implemented this idea for parallel execution of the Omega Test where using DNF is inevitable. Detailed discussion of this procedure could be found in the next section.



(a) Cooper's algorithm (b) The Omega Test

Figure 1: Speedup factors on an 8-core machine

2 Parallelization of the Omega Test

The *Exact Shadow* is an element of the Omega Test which is presented in the form of strict comparisons as follows [4]:

$$\exists\, \mathtt{x}.\ \beta < \mathtt{b}\mathtt{z} \wedge \mathtt{c}\mathtt{z} < \gamma \iff \mathtt{c}\beta + 1 < \mathtt{b}\gamma \tag{2}$$

The *Exact Shadow* has the similar idea to the Fourier-Motzkin elimination for real arithmetic; nevertheless, the discrete property of integer arithmetic requires either $\mathtt{b} = 1$ or $\mathtt{c} = 1$ to proceed the equivalence.

To employ this shadow, we first convert a formula into DNF and apply the shadow until the condition of coefficients is no longer satisfied. Parallelism in our algorithms is evaluated by a language-based parallel cost model, namely the *DAG model of multithreading* [2]. For example, we consider a Presburger instance in the form of a quantified formula with $\mathtt{k}$ quantifiers and a conjunction consisting of $\mathtt{m}$ literals and $\mathtt{n}$ disjunctions of two literals. Transformation of this formula into DNF results in a disjunction of $2^{\mathtt{n}}$ conjunctions of $(\mathtt{m}+\mathtt{n})$ literals. Because these conjunctions are resolved independently based on the rule $\exists\, x_1 \ldots x_n.\ \bigvee_i \bigwedge_j L_{ij} \iff \bigvee_i \exists\, x_1 \ldots x_n.\ \bigwedge_j L_{ij}$, the *parallelism factor* of this example is $\Theta(2^{\mathtt{n}})$. The algorithm contains enough parallelism; therefore, the degree of parallelism is only bounded by the number of used processors. The *DAG model of multithreading* is helpful when it allows us to predict parallel efficiency of an algorithm before proceeding further with implementation.

3 Experimental results

Due to the lack of test suites for Presburger Arithmetic, we attempt to generate some test formulas controllable in terms of size and complexity. Test formulas are formulated by using Pigeon Hole Principle as follows: *given* $\mathtt{N}$ *pigeons and* $\mathtt{K}$ *holes, if* $\mathtt{N} \leq \mathtt{K}$ *there exists a way to assign the pigeons to the holes where no hole has more than one pigeon; otherwise, no such assignment exists.*

Let $\mathtt{x_{ik}}$ be the predicate where pigeon $\mathtt{i}$ is in hole $\mathtt{k}$, and the detailed construction is described below. One important point is that functional programming allows us to express this construction in a natural way which is very close to the logical formalism.

$$\mathtt{P(N,\ K)} = \bigwedge_{\substack{1 \leq i \leq N \\ 1 \leq k \leq K}} (\mathtt{x_{ik}} \Rightarrow \bigwedge_{\substack{1 \leq j \leq N \\ j \neq i}} \neg \mathtt{x_{jk}}) \wedge \bigwedge_{1 \leq \mathtt{i} \leq \mathtt{N}} (\bigvee_{1 \leq \mathtt{j} \leq \mathtt{K}} \mathtt{x_{ij}}) \wedge \bigwedge_{\substack{1 \leq i \leq N \\ 1 \leq k \leq K}} (\mathtt{x_{ik}} \Rightarrow \bigwedge_{\substack{1 \leq j \leq K \\ j \neq k}} \neg \mathtt{x_{ij}}) \tag{3}$$

Satisfiability of Pigeon Hole formulas is known as a provably difficult case for SAT solvers [1]. We define quantified formulas with an arbitrary number of quantifiers in the form of $\exists\, \mathtt{x_{11}} \ldots \mathtt{x_{ab}}.\ \mathtt{P(N,K)}$ where $\mathtt{x_{ik}}$ is encoded as simple equalities, inequalities and divisibility constraints [4]. By doing so, we create quite challenging Presburger instances with predetermined truth values for testing purpose.

We construct the test set for Cooper's algorithm from Pigeon Hole formulas with 21-30 holes, one pigeon and 3 quantifiers for each formula. Each formula is a combination of several independent formulas, which helps to increase degree of parallelism. Results on an 8-core machine are illustrated in Figure 1a showing $4-5\times$ speedup of the parallel version compared to the corresponding sequential one. The results could be improved if one pays more attention to cache usage and minimizes the number of memory allocations.

Another group of test formulas is extracted from Presburger fragments generated by DCMC [5]. We do not describe the details of those formulas, but they have the same form as the example presented in Section 2. Moreover, they consist of many inequalities with very small coefficients (1, -1 or 0) making them become an ideal input for the *Exact Shadow* discussed above.

The test set for the Omega Test consists of formulas extracted from the model-checking problem with 5, 7 and 9 disjunctions respectively. Figure 1b shows speedup factors for the `List-based` and the `Array-based` implementations on the 8-core machine. In general, their speedups are really high with an approximate $5\times$ speedup in the worst case. One should notice that the `Array-based` variant is always more scalable than the `List-based` one. Although the number of cache misses has influence on scalability, the array-based representation is suitable for parallelization due to its advantage of keeping data close together in memory. The result here shows the advantage of using the *Exact Shadow* for alternating quantifiers. Moreover, the parallelization process is fully applicable for the Fourier-Motzkin elimination which is widely used in decision procedures for real numbers.

4 Conclusions

In this paper, we have presented our parallelism concerns regarding decision procedures for PA. A lesson learned is that cache has a huge influence on performance, and even a small change to make data fit in cache could result in $20\times$ performance gain. Moreover, multicore powers are easy to leverage using functional programming when good speedup could be obtained without much parallelization effort. We have achieved good speedups in parallelizing two decision procedures for PA, but the idea should fit the Fourier-Motzkin elimination very well. Although the results may be promising, full exploitation of multicore powers is still an open question for further research.

References

[1] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult SAT instances in the presence of symmetry. In *Proceedings of the 39th annual Design Automation Conference*, 2002.

[2] Guy E. Blelloch. Programming parallel algorithms. *Communications of the ACM*, 1996.

[3] Aaron R. Bradley and Zohar Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 2007.

[4] Phan Anh Dung. Presburger Arithmetic and its use in verification. Master's thesis, Technical University of Denmark, DTU Informatics, 2011.

[5] Michael R. Hansen and Aske W. Brekling. On Tool Support for Duration Calculus on the basis of Presburger Arithmetic. In *18th International Symposium on Temporal Representation and Reasoning*, 2011.

[6] Simon Marlow, Patrick Maier, Hans-Wolfgang Loidl, Mustafa K Aswad, and Phil Trinder. Seq no more: Better strategies for parallel haskell. In *Haskell '10: Proceedings of the Third ACM SIGPLAN Symposium on Haskell*, 2010.

[7] Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger Arithmetic. *Journal of Computer and System Sciences*, 1978.

[8] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Moura. A Concurrent Portfolio Approach to SMT Solving. In *Proceedings of the 21st International Conference on Computer Aided Verification*, 2009.